AU 332 Artificial Intelligence: Principles and Techniques

By: Guowei Deng, Jiangjian Guo

Instructor: Yue Gao

October 13, 2018

I. MINIMAX ALPHA-BETA PRUNING

A. Introduction

In this part, we are going to implement a algorithm to apply in Chinese checkers game. Chinese checkers is a perfect information game for 2 players. A Chinese Checkers board is shown in Figure 1. The goal of the game is to get 10 pegs or marbles from one's starting position to one's ending position as quickly as possible. To win the game, the agent needs to try to make its pegs jump further and prevent the opppsite agent's pegs from jumpping through the board. Our agent use algorithms including minimax iterations, alpha-beta pruning and some other methods.

B. strategy

1. Game start

At the very beginning of the game, the pegs of both side are quite far that they can hardly influence each other. So at that time, we don't need to search a minimax tree to decide which pegs to move first. In this way, we give the agent a static patten which to start the game. The patten just includes 3 steps, and we think after these steps, we need to begin our minimax search to make his way.

2. Minimax alpha-beta pruning

Minimax is the main policy of our Ai. During the game, our Ai choose the move according to a decision-making tree. As we know, every move will generate a different situation, in which there are also many optional moves to make. So, Minimax is to evaluate all the possible states in the next several steps, which is showed nodes in the minimax tree. The important thing is that, to predict the forward situation we need to predict the move that our rival will make.

Assume that: every step we make is the best choice making the situation better for us, however, every step rival makes is the best choice making the situation worse for us(evaluate by the Heuristic function value).Thats why this policy is called Minimax, since we are the max one and the rival is the min one. During the minimax searching, we first generate all the value of the states which are at depth we suppose. For example, if we are doing depth-3 Minimax searching, we need to generate all the value of states which are 3 steps forward from now. After that, Minimax will upload the values. In the max layer, every parent node will choose the action generates the best state. In the mini layer, every parent node will choose the action generates the worse state. The algorithm gets the final move after it reach the root node(a max layer). And that is the move we are going to make.

However, if we take a minute to think of this algorithm, we will find that as we add one layer in the tree, the search space we have a exponential growth. So 2 minutes can be too limited for us to search a deep enough minimax tree. Thats why we need a alpha-beta pruning.

Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves. The general principle is this: consider a node nsomewhere in the tree, such that Player has a choice of moving to that node.If Player has a better choice m either at the parent node of n or at any choice point further up,then n will never be reached in actual play. So once we have found out enough about n (byexamining some of its descendants) to reach this conclusion, we can prune it.

Since minimax is depth-first searching, we can use recursion method to solve it. We need to define two function called max-value and min-value, returning the max or min value of the states generated from current state. As we can imagine, because of the structure of minimax tree, we need to call the functions in a alternant way. Also, when the supposed depth is searched, we don't need to call them anymore.Instead, we call the Heuristic function to get the value of that state and return it. And alpha-beta pruning is easy to achieve, the main point is to set alpha beta and compare them with current value during the search.

3. Heuristic evaluation function

In Chinese checkers game, the agent should try to make its pegs jump further and prevent the opppsite agent's pegs from jumpping through the board. Therefore, in the heuristic evaluation function, we mainly consider these parameters.

- ave_vertical&ave_vertical_op

  These 2 parameters mean the average vertical quantities of both sides. Considering pegs of two sides begin in different position, we make these 2 parameters comparable through a one-to-one mapping. In the meaning time, we give a credit value for pegs in the destination triangle and a punishment value for pegs staying in the beginning triangle.

- variance&variance_hori

  The variance and variance_hori describe the degree of dispersion of our pegs. In order to jump further and faster, agent should't just jump the pegs in the front, leaving the following pegs helpless. So the variance should also be in consideration.

Consequently, the evaluation value can be computed as follow:

$$eval = a * (ave\_vetical - ave\_vertical) - b * \sqrt{variance} - c * \sqrt{variance\_hori} \tag{1}$$

The weight of each parameters can be tuned to fit the game best.

Meanwhile, the heuristic evaluation function also evaluate the position of the state. There are 3 positions we take in consideration. "begin","mid","end" indicate the different situation of the game and in different situation, the agent can perform corresponding reaction. The main criterion to evaluate the situation is the average vertical value. If the value is still small or close to arrival, the position should be "begin" or "end", then the agent will apply functions corresponding to these situations. And if the value is neither small nor close to arrival, which means our pegs are quite related with enemies, the agent uses minimax algorithms.

4. parameters tunning

The main strategy for paremeters tunning is genatic algorithm. Since prof.Gao has presented it in a lecture and the prominent effect of it motivated us to have a try. We first learned more about the basic definitions of GA, and found that it abusolutely may be a good way to adjust the parameters.

It may be too time consuming to write a GA class by our own. So we decided to seek a suitable GA frame in Github and finally chose Gaft, which is lightweight and easy for us to work on. We saw out the examples and became experienced to call the APIs.

After that, we need to change the total framework of rungame into a suitable version for training. So we writed a new py file called train.py, which is especially for GA training. As we know, before game begins, users should press the start button on the board. That makes it impossible for PC to run it on its own. So we deleted it. However, the GUI is still tiresome for training. So we deleted it as well, so that the only thing we get is the result of each round.

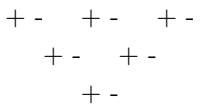
We put the simulate process into the fitness function in GA frame. We just simply input the parameters and the win rate of simulation will be return.It seems to be so good for us to use GA. However the result is not as good as we imagined.

There are several reasons: 1. Every game requires a long time to simulate. Since we need to run at least 10 times to get a reliable win rate, the time of one group of parameters may be at 3 minutes. We have to run dozens of generations and each generation has dozens of individual in the population. The virtual machine can't give us enough calculation power. 2. Our model is always changing, every version need a GA training. That is not practical to get tht best parameters with a so limited caculation power.

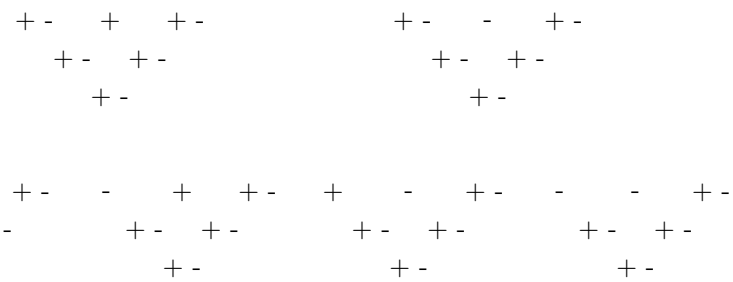
Although using GA seems to be a bad decision in this specific problem, we are sacrisfied that we tried something new and get more familiar with GA. Maybe in the future we will achieve great program by using GA.

II. CSP

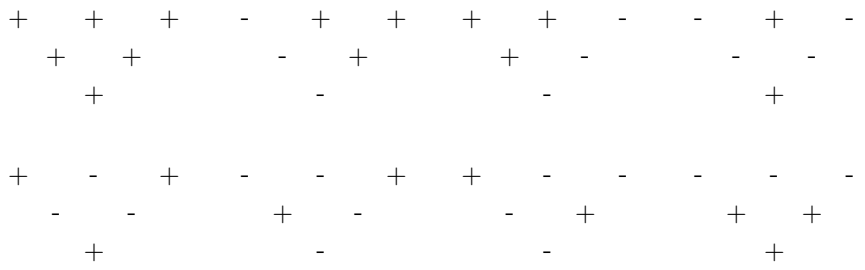
The initial state of the triangle shoule be:



Here is the complete process:



finally



So there are 4 triangles have the same number of "+" and "-".